08

Composición objetos

Transcripción

[00:00] Entonces, ya volviendo a nuestro diagrama, vamos a ordenarlo un poco más. Ya hemos quedado pues que vamos a dejar los implements abajo. Vean. Vamos a subir esto un poco aquí. Listo. Y lo vamos a ordenar. Los implements van a ir aquí abajo y los extends hacia arriba. ¿Por qué?

[00:28] Porque aquí los implements este va a ser nuestro dominio, nuestro dominio de autenticación, y aquí tenemos pues nuestro sistema intermedio, nuestro sistema interno que va a ser digamos el nexo para iniciar sesión en el sistema. Pero a la vez también vamos a tener una clase utilitaria que va a ir por acá.

[00:57] Va a ser AutenticacionUtil. Que esta clase va a estar dentro de cada uno de estos de acá, entonces vamos a poner aquí juntos más o menos, para que se entienda, de estos dos. Perfecto. Entonces, la lógica de autenticación está isolada aquí. La lógica de cómo yo inicio sesión está aquí. Él decide quién entra y quién sale. ¿Quién me marca si soy elegible para entrar o salir? Autenticable.

[01:38] ¿Se dan cuenta cómo automáticamente los conceptos ya salieron, los conceptos ya están separados completamente uno del otro? Cada uno con su propia función, cada uno con su propio trabajo, su propia labor. La implementación de la interfaz me permitió separar ese concepto. Independientemente de eso, mi funcionalidad anterior que yo construí está intacta.

[02:06] Administrador sigue siendo funcionario con todas las características, gerente sigue siendo funcionario con todas las características, y de la misma forma con contador. El único que no se puede autenticar es contador, porque así lo decidió el negocio.

[02:22] Entonces mi sistema está digamos ya hecho, listo para crecer a cualquier escala prácticamente. Este modelo que hemos hecho aquí se puede replicar tranquilamente también en la clase cuenta. ¿Cómo? Si vamos aquí a cuenta podemos pues crear una interfaz.

[02:44] Cuenta aquí está enmarcado como clase abstracta, pero puede ser sí vuelto a una interfaz y que cada uno, por ejemplo acá hay un caso bien curioso, vean acá. La clase cuenta es abstracta y tiene dos hijos que son cuenta de ahorros y cuenta corriente, y cada uno implementa su método SAC y en este caso cobra comisión y cuenta de ahorros no, no cobra nada de comisión.

[03:17] Si nosotros necesitamos personalizar más ese comportamiento podemos poner incluso una interfaz nueva que sea cobrar impuestos o comisión. Con esa interfaz marcamos que cuenta corriente cobra comisión. Entonces sería un extends cuenta implements cobrar comisión. Es un ejemplo más de cómo podemos abstraer eso.

[03:46] Entonces, nuevamente como les voy diciendo, practiquen bastante este concepto de interfaces y polimorfismo y herencia porque es lo que se ve en el día a día de trabajo aquí como desarrollador.